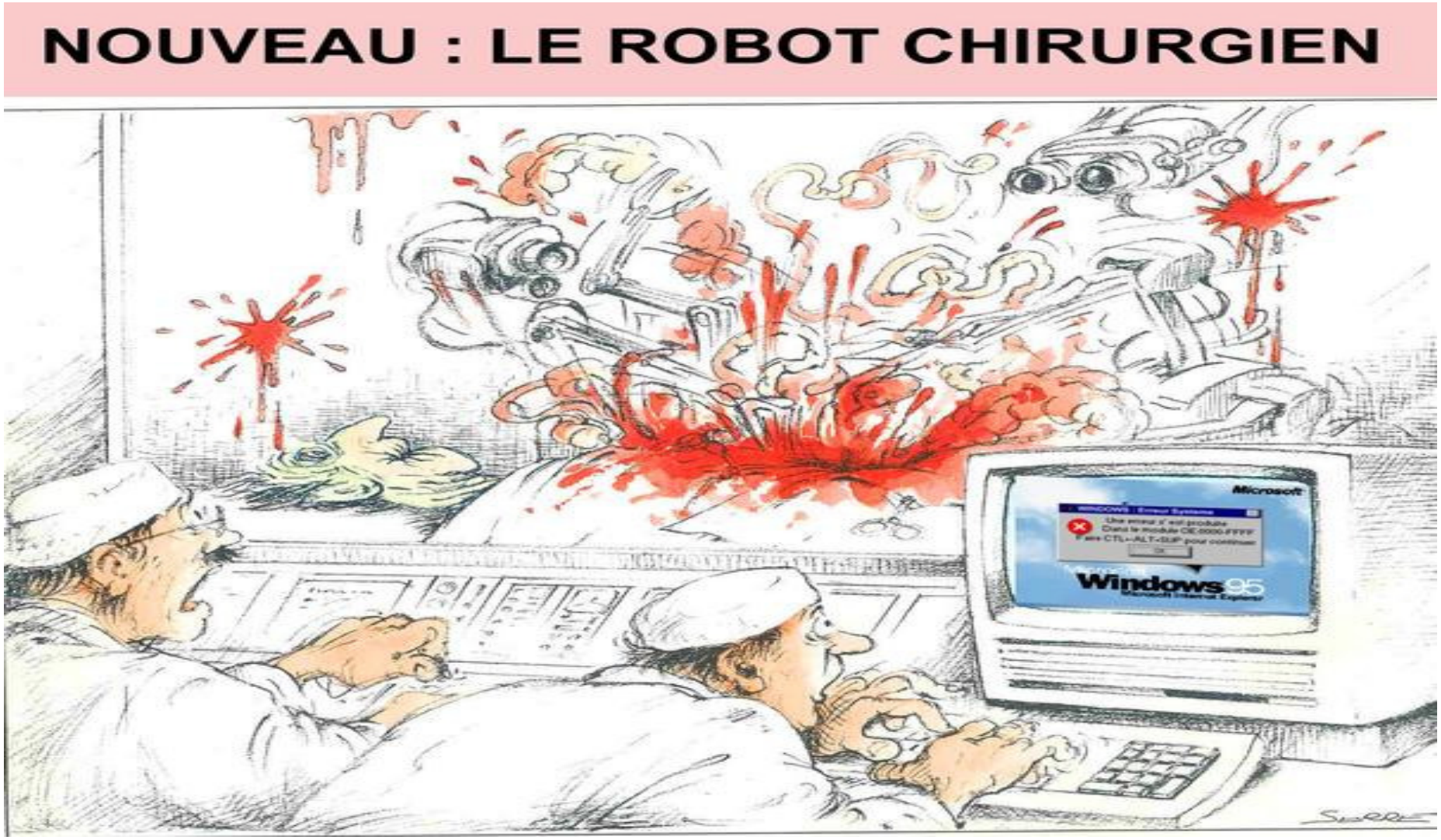
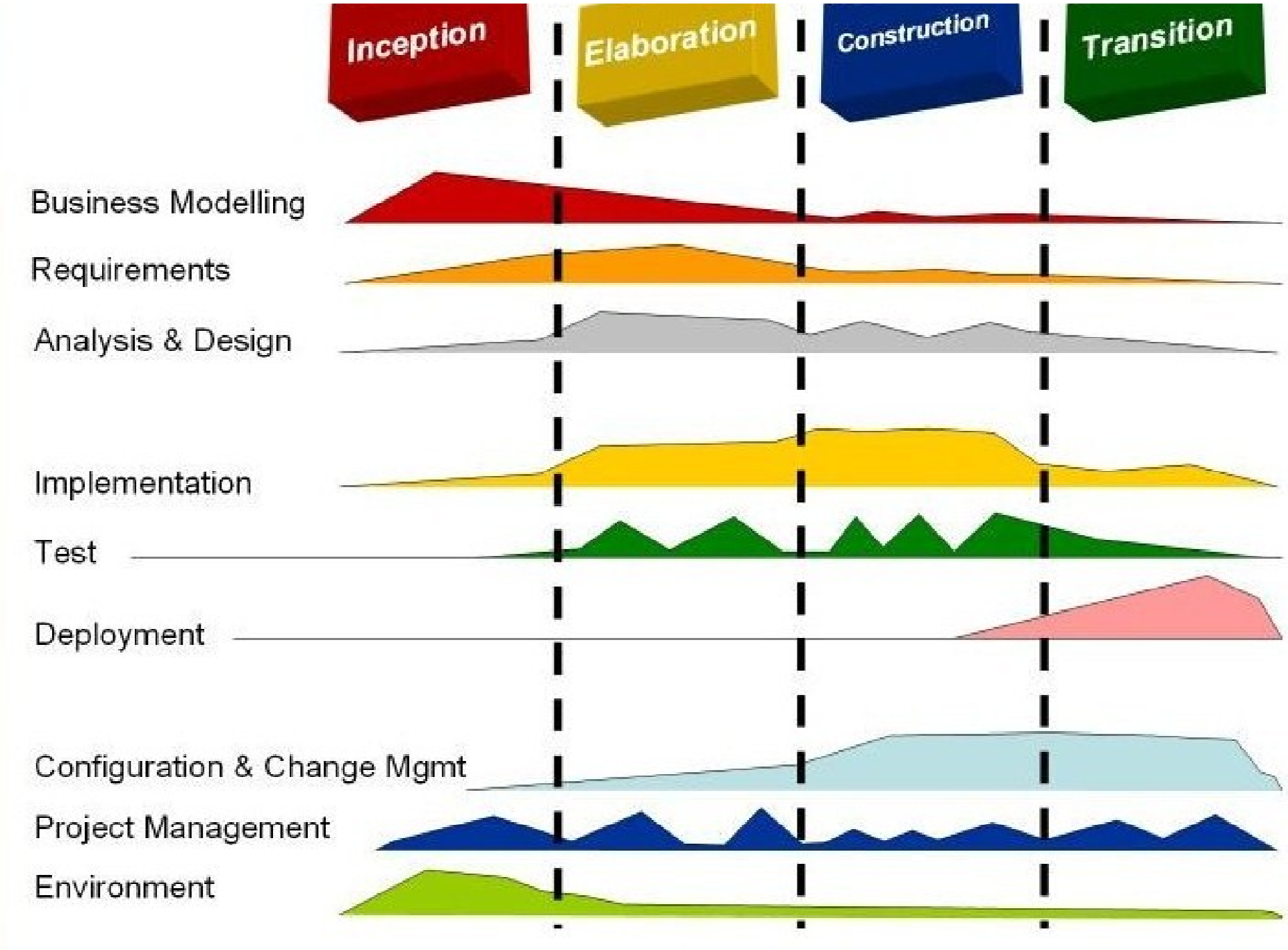
Pruebas

- Pruebas en el PUD
- Las pruebas del software
- Diseño de casos de prueba
  - Tipos de prueba
  - Estrategias de prueba


NOUVEAU : LE ROBOT CHIRURGIEN
Microsoft
Windows 95

Iteración en PUD

- Planificación de la Iteración
- Captura de requisitos:
- Modelo de casos de uso, Modelo de Dominio, ...
- Análisis:
- Diagrama de secuencia del sistema, Contratos, Modelo Conceptual...
- Diseño:
- Diagramas de interacción, Diagrama de Clases
- Implementación:
- codificación (Clases y métodos)
- Pruebas:
- verificación de la implementación
- Evaluación de la iteración



Fuente: http://iseriesvenezuela.blogspot.com/2010/05/rup-y-el-desarrollo-de-software-para.html

Las pruebas de software

- Las pruebas de software son un elemento crítico para la garantía de calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación

- Las pruebas de software son siempre necesarias

- En algunos casos ocupan un 40% del tiempo de un proyecto informático

- Las pruebas pretenden DESCUBRIR ERRORES!

http://www.youtube.com/watch?v=TlMY9j5jh-g&feature=results_main&playnext=1&list=PLF8D60F9C7DE7624A

Las pruebas de software

- Un buen caso de prueba es aquel que tiene una probabilidad muy alta de descubrir un nuevo error

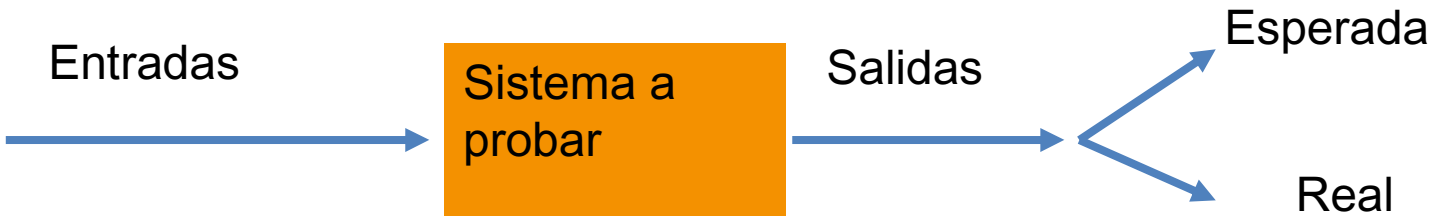
- Una prueba tiene ÉXITO si DESCUBRE un ERROR nuevo

- Debemos diseñar y ejecutar juegos de prueba que, de forma sistemática, detecten distintos tipos de error en el menor tiempo y esfuerzo posible

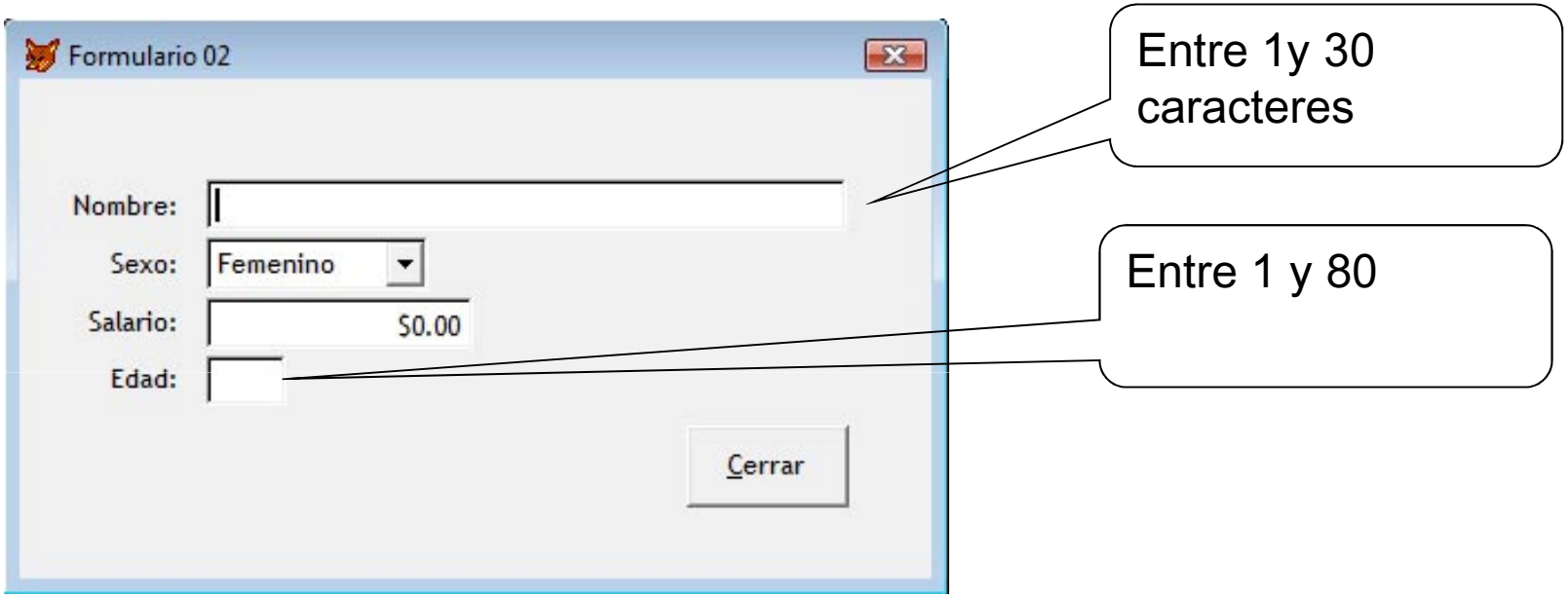
- Los juegos de prueba no deben ser ni demasiado simples ni excesivamente complejos

- Las pruebas PUEDEN DEMOSTRAR la EXISTENCIA de errores, pero NO su AUSENCIA

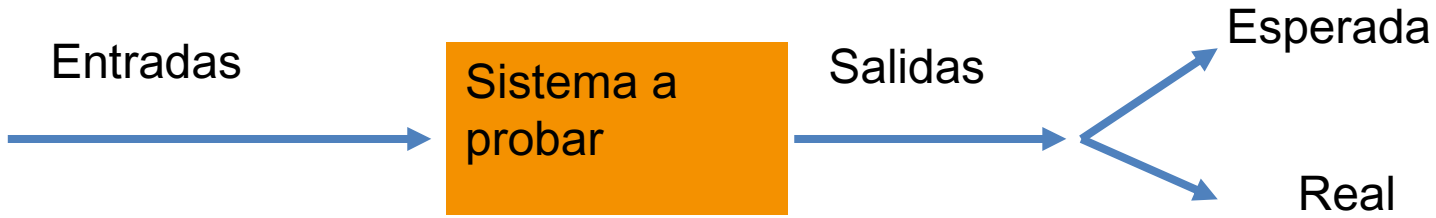
Caso de prueba



Formulario Web



Caso de prueba



Nombre: Raquel Sosa
Sexo: Femenino
Salario: 100
Edad: 23

Caso de prueba 1 → Esperada: Login ok

Nombre: Raquel Sosa
Sexo: Femenino
Salario: 100
Edad: 0

Caso de prueba 2 → Esperada: Login **not** ok

Las pruebas de software

- Las pruebas pueden planificarse mucho antes de que empiecen
- Empezar por lo pequeño y progresar hacia lo grande
- NO son POSIBLE las PRUEBAS EXHAUSTIVAS
- Son más efectivas las pruebas dirigidas por un equipo independiente
- El 80% de los errores está en el 20% de los módulos
- Hay que identificar ésos módulos y probarlos muy bien

Diseño de casos de prueba-Modelo de prueba (PUD)

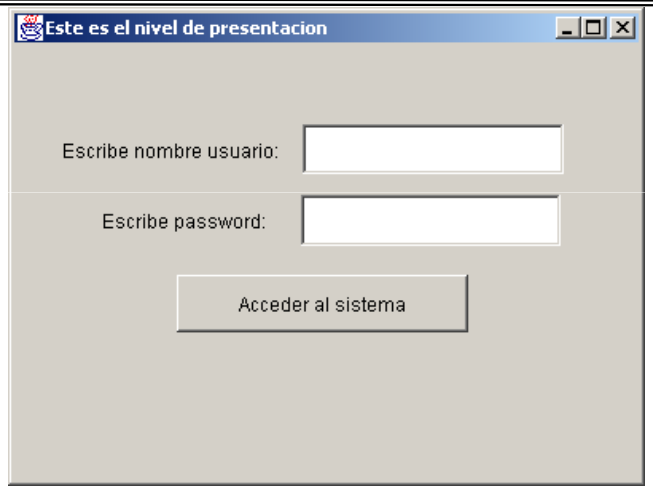
- CASO de prueba
  - Componente que se va a probar
  - Datos de entrada
  - Resultado esperado
  - Condiciones de prueba: estado del componente, información de contexto

- PROCEDIMIENTO de prueba
  - Cómo realizar uno/varios/parte de algún caso de prueba

- COMPONENTE de prueba
  - Automatiza uno/varios/partes de un procedimiento de prueba

Definición: Caso de Prueba

- **Es un conjunto de entradas de prueba, condiciones de ejecución y resultados esperados**
- **Tiene un objetivo concreto (probar algo)**



Ejemplo: CASO de PRUEBA CP1 para
CASO de USO "Entrada Sistema"

ENTRADA: usuario "hacker" password "kaixo"

CONDICIONES DE EJECUCIÓN: no existe en la tabla CUENTA(usuario,pass,intentos) la tupla <"hacker", "kaixo",x> pero sí una tupla <"hacker","hola",x>

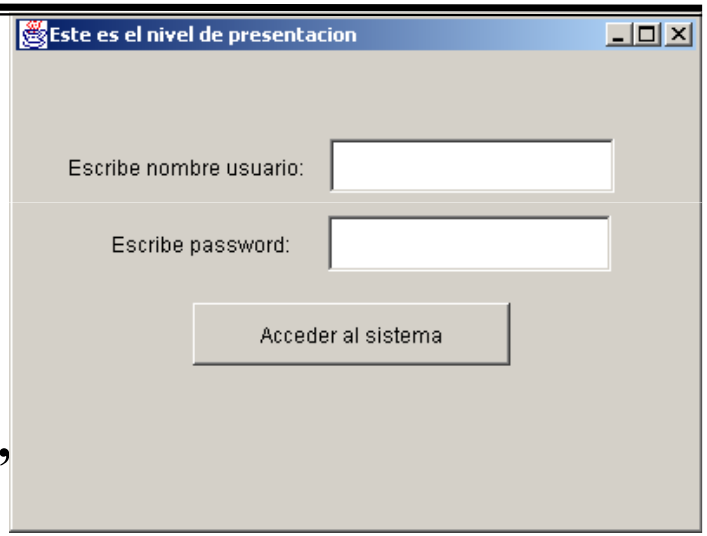
RESULTADO ESPERADO: no deja entrar y cambia la tupla a <"hacker","hola",x+1>

Objetivo del caso de prueba: comprobar que no deja entrar a un usuario existente con un password equivocado.

Definición: Procedimiento de Prueba

- **Pasos que hay que llevar a cabo para probar uno (o varios) casos de prueba: ¿cómo probar el caso de prueba y verificar si ha tenido éxito?**

Ejemplo: Procedimiento de prueba para CP1



- Ejecutar la clase Presentacion

- Comprobar que en la BD "passwords.mdb" existe la tupla <"hacker","hola",x>

- Escribir "hacker"en la interfaz gráfica (en el campo de texto etiquetado "Escribe nombre usuario"

- Escribir "kaixo"en la interfaz gráfica (en el campo de texto "Escribe password")

- Pulsar botón "Acceder al sistema"

- Comprobar que no deja entrar al sistema y que en la BD la tupla ha cambiado a <"hacker","hola",x+1>

Definición: Componente de Prueba

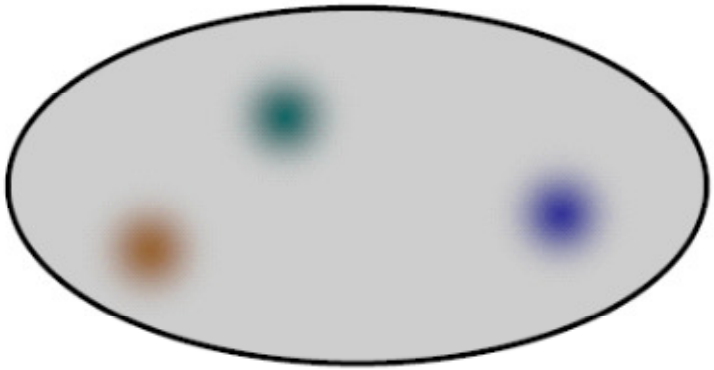
- Programa que automatiza la ejecución de uno (o varios) casos de prueba
- Una vez escrito, se puede probar muchas veces (cada vez que haya un cambio en el código de una clase que pueda afectarle)

```
public class ComponentePruebaEntrSistema {

public void testLoginPassword() {

 InterfaceLogicaNegocio ln; InterfaceOperacionesParaPruebas lp;

 lp.aniadirUsuario("hacker","hola",3); // Crea usuario con pass y numInt.

 boolean b = ln.hacerLogin("hacker","kaixo");

 assertEquals(b,false);

 int j = lp.comprobarUsuario("hacker","hola"); // Dev. Nº intentos

 assertEquals(j,4); //sino error

}
```

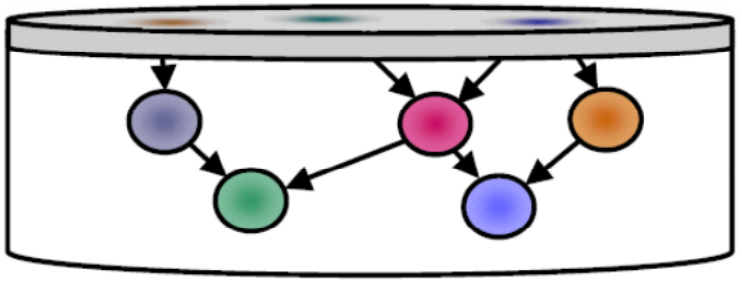
NOTA: se necesitarán otros métodos como **comprobarUsuario,aniadirUsuario** que pueden pertenecer a la lógica del negocio o no (en este caso se considera que no)

Estrategias de Prueba

- Pruebas de Caja Negra
  - Validar si el comportamiento observado del producto cumple sus especificaciones
  - Pruebas funcionales



- Pruebas de Caja Blanca
  - Seleccionar los caminos del programa a ejercitar durante las pruebas
  - Pruebas estructurales



Caja Negra: Clases de Equivalencia

- Dividir el dominio de las entradas en clases de equivalencia
- 2 casos de prueba de una misma partición es probable que revelen los mismos incidentes.

Caja Negra: Clases de Equivalencia

- Identificar las variables y sus posibles valores
- Identificar las clases de equivalencia

Variables	Clases Válidas	Clases Inválidas

- Seleccionar representantes
- Seleccionar casos de prueba

Caja Negra: Clases de Equivalencia

Variables	Clases Válidas	Clases Inválidas	Representantes
Edad	1 a 80	< 1 > 80	60 0 90

Caja Negra: Valores Límite

- Frontera de las particiones: límites
- Mayor probabilidad de revelar fallas
  - Suele haber problemas con los valores límites de los dominios
- Considerar
  - Límite
  - Proximidades al límite
- Son buenos representantes

Caja Negra: Valores Límite

- Modelo de la realidad
  - Derivado de las clases de equivalencia
- Un estrategia de selección
  - Seleccionar valores límite y cercanos
- Un criterio de cobertura
  - Cubrimiento de bordes
- Una teoría de errores
  - Errores por distracción (> por >=)

Caja Negra: Valores Límite

1. Rango
   - La edad es de 1 a 80
   - Límites de la clase válida.
     - Num. art = 1
     - Num. art = 80
   - Límites de las clases inválidas
     - Num. art = 0
     - Num. art = 81

Caja Negra: Valores Límite

Variables	Clases Válidas	Clases Inválidas	Representantes
Edad	1 a 80	< 1 > 80	60 0 90 1 80 0 81 } Valores límite

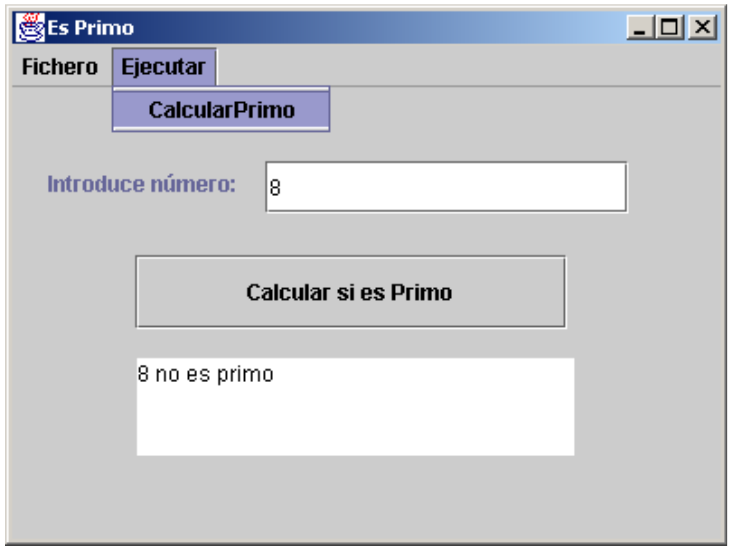
Caja Negra: Resumiendo

- Ventajas
  - Alta probabilidad de detectar incidentes con un conjunto relativamente reducido de casos de prueba
  - Es intuitiva, fácil de aprender y enseñar
  - Es generalizable a múltiples variables
- Limitaciones
  - Errores que no se dan en los bordes o no son los casos más obvios
  - Regresión con los mismos representantes
  - Dominios insuficientemente conocidos

Caja Blanca: "Viendo" el código interno

- Intentan garantizar que todos los caminos de ejecución del programa quedan probados
  - Usa la estructura de control para obtener los casos de prueba.
- Pruebas de estructura de control:
  - De condición: Diseñar casos de prueba para que todas las condiciones del programa se evalúen a cierto/falso
  - De bucles: Diseñar casos de prueba para que se intente ejecutar un bucle 0,1,…,n-1,n y n+1 veces (siendo n el número máximo)

Caja Blanca: Ejemplo




```
public class Esprimo {

  public static boolean esPrimo (String args[])
      throws  ErrorFaltaParametro,
              ErrorSololParametro,
              ErrorNoNumeroPositivo {
      if (args.length == 0) throw new ErrorFaltaParametro();
      else if (args.length > 1) throw new ErrorSololParametro();
      else {
        try {
          float numF = Float.parseFloat(args[0]);
          int num = (int)numF;
          if (num<=0) throw new ErrorNoNumeroPositivo();
          else {
            for (int i=2;i<num;i++)
              if (num%i==0) {return false;}
            return true;
          }
        }
        catch (NumberFormatException e) { throw new ErrorNoNumeroPositivo();  }
      }
  }
```

El método Esprimo.esPrimo puede ser llamado con un array de Strings

Caja Blanca: "Ejemplo

NUM	SalidaEsper	comentario
	falta parámetro	if (args.length == 0) => EJECUTAR "then" // (args.length)==true
xx yy	sólo 1 param	if (args.length > 1) => EJECUTAR "then" // (args.length)==false // (args.length > 1)==true
-4	no positivo	if (num<=0) => EJECUTAR "then" // (args.length > 1)=false // (num<=0)==true
2	primo	for (int i=2;i<num;i++) EJECUTAR BUCLE 0 VECES // (num<=0)==false
3	primo	for (int i=2;i<num;i++) EJECUTAR BUCLE 1 VEZ
4	no primo	for (int i=2;i<num;i++) EJECUTAR 2 VECES // if (num%i==0) => EJECUTAR "then" // (num%i==0)==true
23	primo	for (int i=2;i<num;i++) EJECUTAR BUCLE N VECES // (num%i==0)==false
patata	no positivo	Probar la condición (excepción) catch (NumberFormatException e)

ENTRADA

OBJETIVO A PROBAR

-Probar todas las condiciones

-Probar bucles

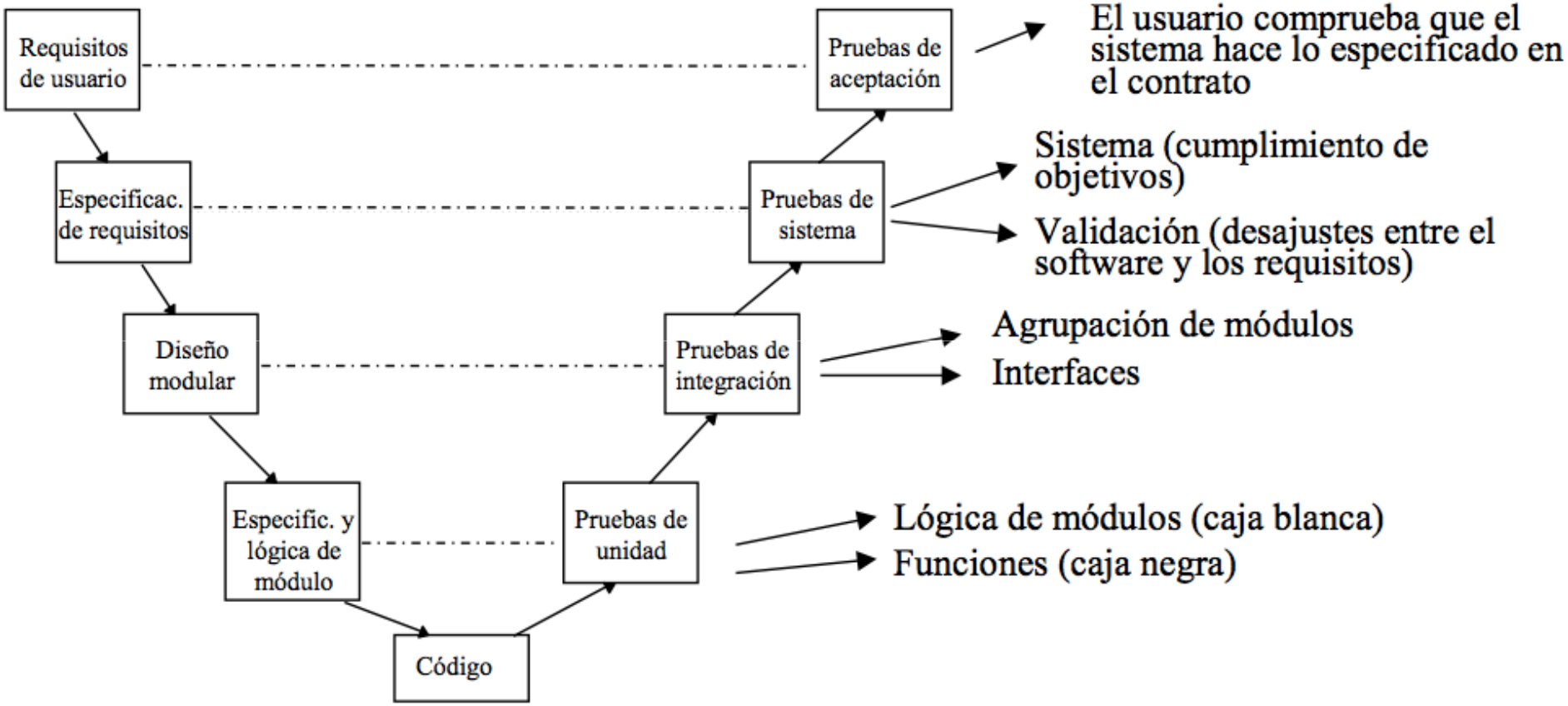
Caja Blanca: Cobertura de decisión

- Todas las decisiones se evalúan al menos una vez (a true Y false).
- Esto garantiza que se evalúen todos los caminos.

Caja Blanca: Cobertura de condición

- En este caso se comprueba que todas las condiciones de las decisiones se evalúan al menos una vez (a cierto o falso).
- Complementarias a las de cobertura de sentencia

¿Qué se puede probar?



Existe una correspondencia entre cada nivel de prueba y el trabajo realizado en cada etapa del desarrollo

Tipos de pruebas

- Pruebas unitarias
  - Prueba de un único comportamiento elemental
- Pruebas de integración
  - Prueba de las interacciones entre componentes del sistema
  - Verificación incremental
    - Descendente
    - Ascendente
- Regresión para detectar errores en componentes ya probados!
- Pruebas del sistema
  - Prueba global del sistema como unidad de ejecución
- Pruebas de aceptación
  - Se centran en asegurar que se satisfacen los requisitos desde el punto de vista del usuario

Tipos de pruebas (otros)

- Pruebas de instalación
  - Verificar que el sistema puede ser instalado en la plataforma del cliente y que funcionará correctamente
- Pruebas de configuración
  - Verificar que el sistema funciona correctamente en diferentes configuraciones (p. ej. Configuraciones de red)
- Pruebas negativas
  - Se centran en provocar intencionadamente que el sistema falle.
  - Se trata de utilizar el sistema en modos para los que no ha sido diseñado: config. de red incorrectas, recursos hw insuficientes, cargas de trabajo imposibles (casos de abuso)
- Pruebas de estrés o tensión
  - Se centran en identificar problemas con el sistema cuando hay recursos insuficientes o competencia por los recursos

Pruebas de Unidad

- Centran la prueba en un componente (1 caso de uso?)
- Puede realizarse en paralelo a otros componentes
- Básicamente son pruebas de caja blanca
  - Interfaz
  - Condiciones límite
  - Caminos independientes
  - Caminos de tratamiento de errores
- Se prueban los caminos de control importantes para descubrir errores en el componente
- Debemos simular el "comportamiento" del resto de componentes

Pruebas de Integración

- Centran la prueba en la integración de varios componentes (casos de uso de un actor?)

- Tipos
  - Pruebas de integración descendente (prog.ppal->modulos)
  - Pruebas de integración ascendente (modulos->prog.ppal)

- Pruebas de regresión
  - Cambios o la introducción de un nuevo componente pueden provocar errores en componentes ya probados!
  - Al realizar cambios en algún componente debemos probar de nuevo los componentes ya probados
  - Se realizan las mismas pruebas para asegurarse que no se han producido cambios colaterales

Pruebas de Sistema

- Realizado el software, éste debe ponerse en explotación e integrarse en un entorno productivo

- Estas pruebas sirven para verificar que se han integrado adecuadamente todos los elementos del sistema y todos ellos de forma conjunta realizan las funciones apropiadas

- Pruebas de seguridad
- Pruebas de resistencia
- Pruebas de rendimiento (carga)
- Pruebas de recuperación

Pruebas de Aceptación

- Se llevan a cabo cuando se han terminado las pruebas de integración, el software está ensamblado y se han realizado todas las pruebas de unidad e integración
- La validación se consigue cuando el software funciona según las EXPECTATIVAS del USUARIO
- Se realizan una serie de pruebas de caja negra que aseguren que se satisfacen los requisitos

- Funcionales
- De rendimiento
- De documentación
- Recuperación de errores
- ...

Pruebas de Aceptación

- Pruebas ALFA:

- Realizadas por el usuario con el desarrollador como observador en un entorno controlado (simulación de un entorno de producción)

- Pruebas BETA:

- Realizadas por el usuario en su entorno de trabajo y sin observadores

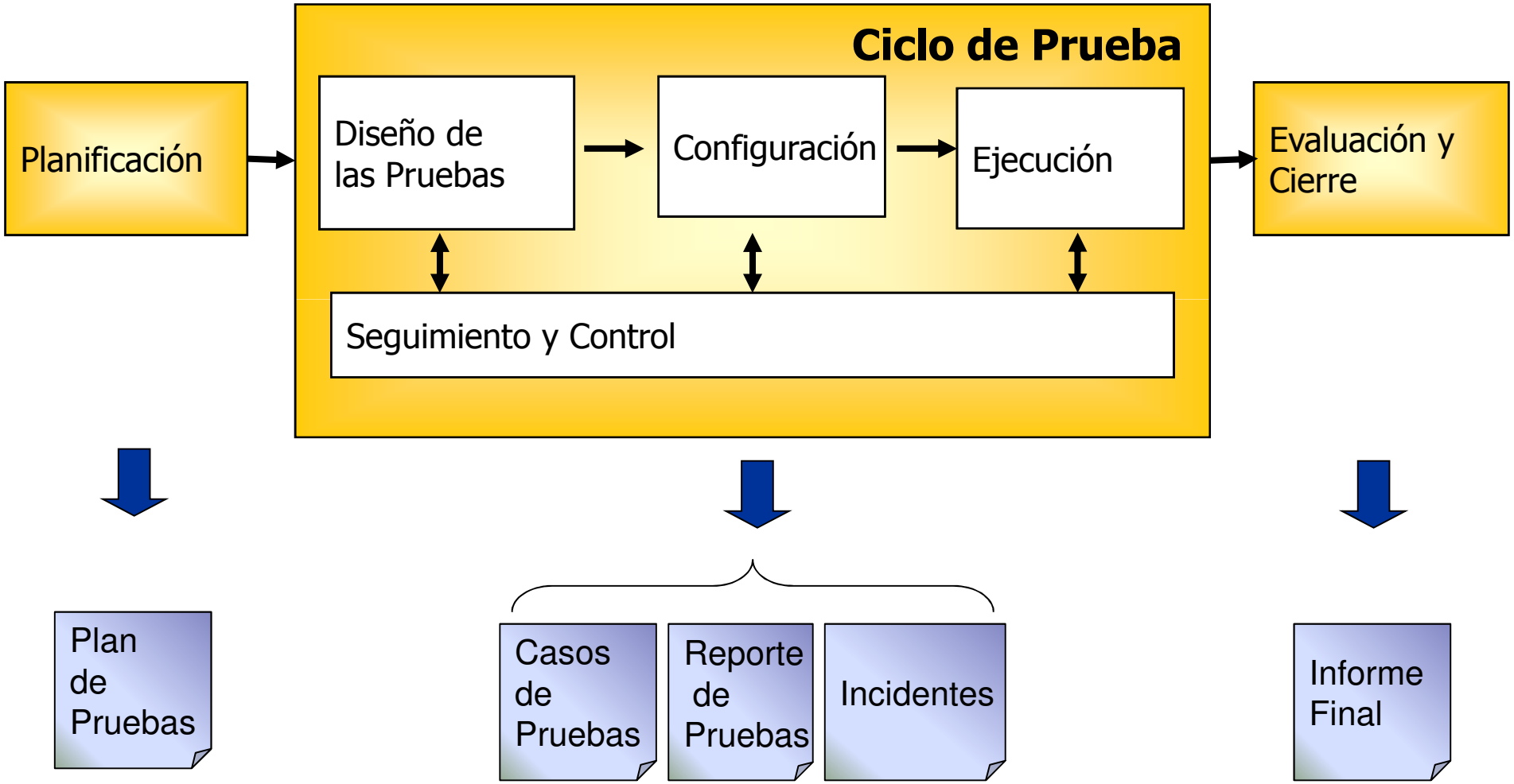
Depuración de errores

- Al realizar pruebas pueden descubrirse errores y éstos deben depurarse

- Depurar errores es extremadamente DIFÍCIL (sobretodo si se trata de un sistema desconocido)

- El error puede ser provocado por un "mal uso" no contemplado en el diseño
- Puede ser difícil reproducir las condiciones que lo producen
- El error aparece de forma intermitente
- Su corrección requiere cambios sustanciales del SI

Proceso de Pruebas



Pero yo soy programador…

- La calidad del software final depende de todos los involucrados
- Cuanto antes se encuentra un error, menor es el coste de arreglarlo
- Existen herramientas que ayudan a los programadores a probar su código

Caso de prueba (para la práctica)

- CÓMO debe ser
  - Con alta probabilidad de detectar algún error
  - No redundante
  - Representativo
  - Ni muy simple ni muy complejo

- QUÉ debe contemplar
  - La planificación de la prueba
  - El diseño de los casos de prueba
  - La ejecución de la prueba
  - La evaluación de los casos de prueba-cobertura del CP y estado de los defectos

Caso de prueba (para la práctica)

- No hay que olvidarse de:

- Pruebas sobre la capa de presentación (ventanas, menús, ratón...)

- Pruebas sobre la capa de gestión de datos

- Pruebas de documentación-manuales

- Pruebas de ayuda